USING LINEAR PROGRAMMING AS A SIMPLEX SUBROUTINE

R. J. Clasen

November 1965

P-3267

ABSTRACT

This Paper discusses the problems involved in using linear programming as a subroutine of a larger routine. Proposals are made for eliminating the tolerance selection problem, and for improving the accuracy of inversions.

Sample programs are given in FORTRAN IV and in ALGOL.

USING LINEAR PROGRAMMING AS A SIMPLEX SUBROUTINE

R. J. Clasen[+]

The RAND Corporation, Santa Monica, California

INTRODUCTION

Use of linear programming has increased rapidly in recent years. The major use was and still is to solve a problem stated in terms of linear programming; generally, only the answer to the problem was necessary. The one difficulty, usually, was preparing the input in a suitable format, and obtaining the output answer in an equally suitable format.

However, a problem occasionally arises of which only a small part involves obtaining the answer to a linear programming problem. For example, one may use linear programming to initially estimate the solution to a chemical equilibrium problem [1]. In this case, it would be convenient to have linear programming as a subroutine. We

---

[+]Any views expressed in this paper are those of the author. They should not be interpreted as reflecting the views of The RAND Corporation or the official opinion or policy of any of its governmental or private research sponsors. Papers are reproduced by The RAND Corporation as a courtesy to members of its staff.

define our linear programming problem as that of determining $x_1, x_2, \ldots, x_n$ such that

$$\phi = \sum_{j=1}^{n} c_j x_j$$

is a minimum over all sets $x_1, x_2, \ldots, x_n$ that satisfy

$$\sum_{j=1}^{n} a_{ij} x_j = b_i \qquad i=1,2,\ldots,m ,$$

and

$$x_j \geq 0 \qquad j=1,2,\ldots,n .$$

The details of the construction of such a subroutine are the subject of this Paper. The criteria for determining what procedures should be used are:

1) Accuracy of the solution, including its independence of row and column scale factors,
2) Storage needed for data,
3) Length of the subroutine needed,
4) Speed of solution--

in that order.

DETERMINING TOLERANCES

A person solving linear programming problems with the simplex method [2] generally has trouble estimating certain

tolerances. Typically, the following three tolerances are needed:

1) The Pivot Tolerance: a number in the pivot column is considered zero if its absolute value does not exceed the pivot tolerance.
2) The Zero Tolerance: a number in the solution vector is considered zero if its absolute value does not exceed the zero tolerance.
3) The Cost Tolerance: a reduced cost is considered zero if its absolute value does not exceed the cost tolerance.

We abbreviate these tolerances as TP, TZ, and TC, respectively.

When using a simplex subroutine, the user is unable to see his input numbers before he uses the subroutine; hence, he is not able to determine the proper tolerances. Therefore, the simplex subroutine should either calculate its own tolerances, or use a method that does not need tolerances. We used a combination of these two procedures.

Before giving the procedure for obtaining tolerances, let us first review the basic steps in the simplex method using the "explicit inverse" method:

1) Check the solution vector for feasibility. Let the basic part of the solution vector be denoted $w_1, w_2, \ldots, w_m$.†

---

†This is what Cutler and Wolfe [3] call $x_{j_1}, x_{j_2}, \ldots, x_{j_m}$.

2) Calculate the prices--the "phase one" prices if the problem is not yet feasible, the "phase two" prices if the problem is feasible.

3) Calculate the reduced costs and find the column, JT, with the minimum reduced cost, MRC. If MRC $\geq$ 0, terminate the subroutine; if MRC $<$ 0, the pivot column is column JT.

4) Obtain the column vector JT by multiplying the inverse and the original column JT. Let the column obtained be called $y_1, y_2, \ldots, y_m$.

5) Obtain the pivot row, IR, by using the subscript that causes the quantity $\frac{w_i}{y_i}$ to be a minimum for all non-zero y_i for which $\frac{w_i}{y_i} \geq 0$. Slight variations of this rule may be used when $w_i < 0$. If no row is found, then for each i either $y_i = 0$ or $\frac{w_i}{y_i} < 0$. In this case, we have an infinite solution and the subroutine is terminated.

6) Update the inverse, the "phase two" prices, and the w_i by pivoting on (IR, JT).

These steps are typically repeated until the subroutine terminates at either Step 3 or Step 5. The initial basis may be vacuous (or "artificial") and the initial inverse may be the identity. In addition to these steps, every m/2 to m iterations, it is usually desirable to "re-invert" the basis, so that the round-off error is not too large. If the re-inversion is to be done every NVER times, we adjoin to this system a counter, INVC (which is zero initially), and the following step:

7) Increase INVC by 1. If INVC < NVER, go to Step 1. Otherwise set INVC to zero and invert the basis, then go to Step 1.

The pivot tolerance (TP) is used in Step 5 to determine whether or not a y_i is zero. A satisfactory method of computing this tolerance is first to compute, on every iteration, YMAX = $\max_{i=1}^{m} |y_i|$. Then the pivot tolerance _for that iteration_ is taken to be TP = YMAX*2^{-16}. Then, on a machine that carries numbers in floating point notation to a relative accuracy of 2^{-27}, we assume $y_i = 0$ in Step 5 if $|y_i| \le$ TP. Thus we are, in effect, assuming the last 11 bits are not significant. While this is an _ad hoc_ rule, it has worked reasonably well for problems on the order of 50 constraints.

The zero tolerance (TZ) is typically used in Step 1 to eliminate small negative numbers when determining feasibility. We do not calculate a zero tolerance, but we eliminate the small negative numbers _at the source_, i.e., in Step 6--the pivot. In other words, when we calculate a new solution vector w_i', we keep the old solution vector w_i; then, if $w_i \ge 0$ and $w_i' < 0$, we _set_ w_i' to zero. These negative w_i can also arise after a reinversion. If the reinversion produces a full basis, and the problem was feasible before the reinversion began, we set to zero any negative w_i generated by the inversion.

Step 3 requires the cost tolerance (TC) to determine whether a small negative cost is "really" zero. "Really" zero means the number would be zero if infinite computing accuracy were used. The present method is to temporarily ignore the possibility that a reduced cost is a rounded zero, and to find the minimum reduced cost using no (or a zero) tolerance. Then we proceed as usual to Step 4 to obtain the prospective pivot column and the pivot tolerance. We use the number TP for TC, but we do not terminate the first time the minimum reduced cost exceeds -TC. Instead, we require the minimum reduced cost to exceed -TC for two consecutive iterations.

Now, if the column to be pivoted on "really" had a zero reduced cost, we may find no pivot row in Step 5. If we do find a pivot row in a column with a reduced cost of zero, it does no harm (except possibly some lost time) to pivot in this column, since we will merely obtain an alternate optimal solution. If in Step 5 we find no pivot row, we do not declare an infinite solution unless the problem is feasible, and the minimum reduced cost (MRC) calculated in Step 3 satisfies: $-1000 * \text{MRC} < \text{YMAX}$. If these conditions are not satisfied, we declare an optimal solution if the problem is in a feasible state, or a "no feasible

solution" if the problem is in an infeasible state. This additional test effectively prevents a false declaration of an "infinite solution."

Some linear programming routines allow redundant constraints. Because we use an effective zero tolerance of zero, we do not allow redundant rows. Hence if we are not able to pivot in every row, we declare an infeasible solution. Thus, a problem is infeasible if there exist numbers $p_1, p_2, \ldots, p_m$ (m being the number of constraints) not all zero such that

$$\sum_{i=1}^{m} p_i a_{ij} \le 0 \quad \text{for} \quad j=1,2,3,\ldots,n$$

and

$$\sum_{i=1}^{m} p_i b_i \ge 0 \ .$$

If we have no feasible solution to a problem, the shadow prices obtained will be proportional to the above p_i. (The proportion may be the negative of the p_i above, in which case both inequalities above would be reversed.)

INVERSION METHOD

The method of inverting in our subroutine differs from the usual procedure. A typical method of inverting is to pivot in each column that is part of the basis, pivoting in the row in which the column entry has the largest absolute value. For example, suppose

$$A = \begin{pmatrix} 2 & 1 \\ 1 & 0 \end{pmatrix}, \qquad B = \begin{pmatrix} 1 \\ \epsilon \end{pmatrix}$$

where ϵ is a small number whose absolute value is less than half the accuracy to which unity can be represented. Thus, on an IBM 7090, ϵ might be 10^{-9}. Now pivoting in the first column, we would chose the first row and obtain:

$$\bar{A} = \begin{pmatrix} 1 & \frac{1}{2} \\ 0 & -\frac{1}{2} \end{pmatrix}, \qquad \bar{B} = \begin{pmatrix} \frac{1}{2} \\ \epsilon-\frac{1}{2} \end{pmatrix}.$$

But we assume that ϵ is small enough so that $\epsilon-\frac{1}{2}$ numerically becomes $-\frac{1}{2}$. Then we pivot in the second row of the second column and obtain:

$$\bar{A} = I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \qquad \bar{B} = W = \begin{pmatrix} 0 \\ 1 \end{pmatrix};$$

hence, $w_1 = 0$, $w_2 = 1$. We note the correct answer is $w_1 = \epsilon$, $w_2 = 1-2\epsilon$. The ϵ has been absorbed into the round-off error.

The following pivot scheme avoids this problem:

1) Let $y_1, y_2, \ldots, y_m$ be the transformed column in which we wish to pivot. Let $\bar{b}_1, \bar{b}_2, \ldots, \bar{b}_m$ be the current values of the transformed constant vector (i.e., what was previously called w_i).
2) Let S be the set of integers such that $i \epsilon S$ if $|y_i| > TP$ where TP is the pivot tolerance calculated as above.
3) Let T be the set of integers such that $i \epsilon T$ if $\bar{b}_i = 0$.
4) If $S \cap T$ is not vacuous, do Step A; if $S \cap T$ is vacuous, do Step B.

   A) Let IR, $IR \epsilon\ S \cap T$, be an integer such that $|y_{IR}| \geq |y_i|$ for all $i \epsilon\ S \cap T$.

   B) Let IR, $IR \epsilon S$, be an integer such that

$$\left| \frac{y_{IR}}{\bar{b}_{IR}} \right| \geq \left| \frac{y_i}{\bar{b}_i} \right| \quad \text{for all } i \epsilon S.$$

Thus we are effectively pivoting in the row that has the largest ratio $|y_i/\bar{b}_i|$. In the above example, the largest ratio in the first column is the second row; hence the result of the first pivot is

$$\bar{A} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \qquad \bar{B} = \begin{pmatrix} 1-2\epsilon \\ \epsilon \end{pmatrix}.$$

Here the $1-2\epsilon$ will be rounded, presumably, to unity. Then we pivot in the first row of the second column to obtain:

$$\bar{A} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \qquad \bar{B} = \begin{pmatrix} 1 \\ \epsilon \end{pmatrix}.$$

Hence we get $w_2 = 1$, $w_1 = \epsilon$, which is actually the correct answer within round-off error. This method has the disadvantage that, in not choosing the largest element in a column to determine the pivot row, we may get an element that barely exceeds the pivot tolerance. But we nonetheless believe that the increased accuracy justifies this approach.

Other choices could have been made for the pivot element. For example, instead of pivoting on columns in column order, one could first choose the eligible row with the smallest w_j, then choose from that row of the matrix the elegible element with the largest absolute value. This was not done because of timing considerations, and because it would make the selections scale-dependent. Note that, in our method, the only scale-dependent operation is finding the pivot column in Step 3.

The simplex subroutine described here is available[+] as a FORTRAN IV routine through the IBM-users group (SHARE). The cards for this subroutine may be obtained by writing directly to

SHARE Distribution Agency
DP Program Information Department
IBM Corporation
40 Saw Mill River Road
Hawthorne, New York 10532

and asking for SHARE Distribution Agency No. 3384.

[+] Employees of RAND may obtain a copy from the SHARE program librarian, Pearl Leonhardt, by asking for the W026 routine.

Appendix

THE FORTRAN SUBROUTINE

The simplex subroutine, SIMPLE, may be used to solve a general linear programming problem of the form: Find $x_1, x_2, x_3, \ldots, x_n$, such that

$$\sum_{j=1}^{n} C_j x_j \tag{1}$$

is a minimum over all sets $x_1, x_2, x_3, \ldots, x_n$ that satisfy the independent constraints:

$$\sum_{j=1}^{n} a_{ij} x_j = b_i \qquad i=1,2,3,\ldots,m\ , \tag{2}$$

and $x_j \geq 0$ $j=1,2,\ldots,n$, where C_j, a_{ij}, b_i are fixed numbers.

USAGE

The a_{ij} is stored in a two-dimensional array, A, with a_{ij} in cell A(i,j); C_j is stored in a one-dimensional array, C, with C_j in cell C(j); and b_i is stored in a one-dimensional array, B, with b_i in cell B(i).

The calling sequence is

CALL SIMPLE (II,M,N,A,B,C,KO,X,P,JH,XX,Y,PE,E)

where

II =	0;
M =	Number of rows, m;
N =	Number of variables, n;
A, B, C	are as above;
KO =	A subscripted variable of dimension 6;
X =	A subscripted variable of dimension n or more;
P, JH, XX, Y, and PE =	Subscripted variables of dimension m or more; and
E =	A subscripted variable of dimension m^2 or more.

The dimension of A (line 0008 of FORTRAN listing) must agree (at least in the first subscript) with the dimension of A in the calling program. The other dimensions need not agree with those of the calling program. The subroutine does not change II, M, N, A, B, or C.

RESULTS

Upon exiting from the subroutine,

X(1),X(2),...,X(n) contains $x_1, x_2, \ldots, x_n$ (the solution);

P(1),P(2),...,P(m) contains the shadow prices, i.e., the negative of the dual solution;

KO(1) contains an 0 if the problem was feasible, 1 if the problem was infeasible, 2 if the problem had an infinite solution, and 4 or 5 if the algorithm did not terminate;

KO(2) is the number of iterations taken;

KO(3) is the number of pivots performed since the last inversion;

KO(4) is the number of inversions performed;

KO(5) is the number of pivot steps performed;

and

KO(6) contains, if the problem has an infinite solution, the number of the variable that was infinite.

If an initial basis is available, this basis may be communicated to the subroutine by letting

$$\mathrm{II} = 1\ ,$$

$$X(i) = \begin{cases} 0.0 \text{ if variable } i \text{ is not in basis,} \\ \text{(non-zero) if variable } i \text{ is in basis,} \end{cases}$$

and the other quantities remain as above.

If the constraints (2) are linearly dependent, the problem is considered infeasible. If the problem is infeasible, (KO(1) = 1), the P(i) contains the coefficients

that generate the infeasible row; i.e., if the problem is infeasible, the infeasibility is represented by

$$\sum_{j=1}^{n} D(j)X(j \qquad {}^{m} \; P(i)B(i) = E$$

where

$$D(j) = \sum_{i=1}^{m} P(i)A(i,j) \qquad j=1,2,\ldots,n \; .$$

Then, when infeasible, D(1),D(2),...,D(n), and -E will simultaneously all be either non-negative or non-positive; and if E is zero, then D(1),D(2),...,D(n) are all zero.

If the problem has an infinite solution, (KO(1) = 2), define a vector X1 as follows:

```
  DØ 1  J=1,N
1 X1(J) = 0.
  DØ 2  I=1,M
    J= JH(I)
2   X1(J) = -Y(J)
  J = KO(6)
  X1(J) = 1.
```

Then the infinite solution is represented by

$$\lim_{t \to \infty} \quad (X(J) + t \cdot X1(J)) \; .$$

If the problem did not terminate, then 4m+10 iterations were insufficient to complete the solution. If KO(1) = 5, the problem is not yet feasible; and if KO(1) = 4, the problem is feasible but not yet optimal. The subroutine may be called a second time, with II=1, or the equation on card 0021 may be changed to allow for more iterations.

EXAMPLE

Let

$$\begin{aligned} 3x_1 \qquad\qquad &+ 14x_4 + 1x_5 + 1x_6 = 7 \,, \\ 1x_2 \qquad &+ 16x_4 + \tfrac{1}{2}x_5 - 2x_6 = 5 \,, \\ 1x_3 &+ 1x_4 \qquad\qquad\qquad = 0 \,, \end{aligned}$$

and minimize: $- 28x_4 - x_5 - 2x_6$.

Using the subroutine with DIMENSION A(50,136), we zero out A and set

A(1,1) = 3.	C(1) = 0.
A(2,2) = 1.	C(2) = 0.
A(3,3) = 1.	C(3) = 0.
A(1,4) = 14.	C(4) = -28 .
A(2,4) = 16.	
A(3,4) = 1.	
A(1,5) = 1.	C(5) = -1 .
A(2,5) = 0.5	
A(3,5) = 1.	
A(1,6) = 1.	C(6) = -2 .
A(2,6) = -2.	
B(1) = 7.	
B(2) = 5.	
B(3) = 0.	
M = 3	
N = 6	
II = 0	

Then we

```
CALL SIMPLE (II,M,N,A,B,C,KO,X,P,JH,XX,Y,PE,E)
```

where B,P,JH,XX,Y, and PE have DIMENSION of at least 3 (i.e., M), KO has DIMENSION 6, C and X have DIMENSION of at least 6 (i.e., N), and E has DIMENSION of at least 9 (i.e., M^2).

Upon returning from the subroutine, we find the following values in storage:

```
KO(1) = 0   Optimum solution,
KO(2) = 2   Number of iterations taken,
KO(3) = 2   Number of pivots since last inversion,
KO(4) = 1   Number of inversions performed,
KO(5) = 5   Total number of pivots,
KO(6) = 0   Not applicable.
```

Also

$P(1) = 2.$
$P(2) = 0.$
$P(3) = 2.38 \times 10^{-7} \cong 0.$

and

$X(1) = 0.$
$X(2) = 18.9999998 \cong 19.$
$X(3) = 0.$
$X(4) = -0. = 0.$
$X(5) = 0.$
$X(6) = 6.99999994 \cong 7.$

Now we may calculate the reduced costs from this information using, for example, the following instructions to put the $j^{\underline{th}}$ reduced cost into CBAR(j):

```
      DO  12  J = 1,N
      CBAR(J) = C(J)
      DO  11  I = 1,M
   11   CBAR(J) = CBAR(J) + P(I)*A(I,J)
   12 CONTINUE
```

This results in

$$
\begin{aligned}
CBAR(1) &= 6. \\
CBAR(2) &= 0. \\
CBAR(3) &= 2.384 \times 10^{-7} \simeq 0. \\
CBAR(4) &= 2.384 \times 10^{-7} \cong 0. \\
CBAR(5) &= 1. \\
CBAR(6) &= 0.
\end{aligned}
$$

Following is a listing of the FORTRAN program written in FORTRAN IV [4], then an ALGOL procedure using McCracken [5]. The ALGOL procedure is a translation of the FORTRAN program, so any discrepancy should be resolved in favor of the FORTRAN program. Table 1 gives the names and meanings of the symbols used in the calling sequence to the FORTRAN program, in the FORTRAN program itself, and in the ALGOL procedure.

Table 1

SYMBOL NAMES AND MEANINGS

FORTRAN Program		ALGOL Procedure (Lower Case)	Usage
Calling Sequence	Subroutine		
II	INFLAG	INFLAG	Input flag denoting whether a basis is specified.
M	MX	M	Number of constraints.
N	NN	N	Number of variables.
A	A	A	Constraint matrix.
B	B	B	Right hand side.
C	C	C	Cost vector.
KO	KO	KO	Output parameters.
(See below)[a]	KB	KB	List of rows pivoted on (by column).
P	P	P	Shadow prices.
JH	JH	JH	List of columns pivoted on (by row).
XX	X	X	Basis part of solution vectors $(w_1, w_2, \ldots)$.
Y	Y	Y	Last pivot column.
PE	PE	PE	Negative of dual solution (sa[illegible]s P when feasible).
E	E	E	Inverse of basis: element e_{ij} is stored in E(i+m·(j-1)).
X	(See below)[a]	Z	Full solution vector.

[a]In the FORTRAN subroutine, the quantities corresponding to the ALGOL symbols Z and KB are stored in the same location, but at different stages of the algorithm solution.

FORTRAN PROGRAM

```
$IBFTC SIMPLE  REF                                                       LSUB0001
C  AUTOMATIC SIMPLEX      REDUNDANT EQUATIONS CAUSE INFEASIBILITY        LSUB0002
      SUBROUTINE SIMPLE(INFLAG,MX,NN,A,B,C,KO,KB,P,JH,X,Y,PE,E)          LSUB0003
      REAL B(1),C(1),P(1),X(1),Y(1),PE(1),E(1)                           LSUB0004
      INTEGER INFLAG,MX,NN,KO(6),KB(1),JH(1)                             LSUB0005
      EQUIVALENCE (XX,LL)                                                LSUB0006
C THE FOLLOWING DIMENSION SHOULD BE THE SAME HERE AS IT IS IN CALLER.   LSUB0007
      REAL A(50,136)                                                     LSUB0008
      REAL AA,AIJT,BB,COST,DT,RCOST,TEXP,TPIV,TY,XOLD,XX,XY,YI,YMAX      LSUB0009
      INTEGER I,IA,INVC,IR,ITER,J,JT,K,KBJ,L,LL,M,M2,MM,N                LSUB0010
      INTEGER NCUT,NPIV,NUMVR,NVER                                       LSUB0011
      LOGICAL FEAS,VER,NEG,TRIG,KQ,ABSC                                  LSUB0012
C                                                                        LSUB0013
C         SET INITIAL VALUES, SET CONSTANT VALUES                        LSUB0014
      ITER = 0                                                           LSUB0015
      NUMVR = 0                                                          LSUB0016
      NUMPV = 0                                                          LSUB0017
      M = MX                                                             LSUB0018
      N = NN                                                             LSUB0019
      TEXP  =  .5**16                                                    LSUB0020
      NCUT  =  4*M + 10                                                  LSUB0021
      NVER  =  M/2  +  5                                                 LSUB0022
      M2 = M**2                                                          LSUB0023
      FEAS = .FALSE.                                                     LSUB0024
      IF  (INFLAG.NE.0)  GO TO 1400                                      LSUB0025
C* 'NEW'    START  PHASE ONE WITH SINGLETON BASIS                        LSUB0026
      DO 1402  J = 1,N                                                   LSUB0027
        KB(J) = 0                                                        LSUB0028
        KQ = .FALSE.                                                     LSUB0029
        DO 1403  I = 1,M                                                 LSUB0030
          IF (A(I,J).EQ.0.0)  GO TO 1403                                 LSUB0031
          IF (KQ.OR.A(I,J).LT.0.0) GO TO 1402                            LSUB0032
          KQ = .TRUE.                                                    LSUB0033
 1403   CONTINUE                                                         LSUB0034
        KB(J) = 1                                                        LSUB0035
 1402 CONTINUE                                                           LSUB0036
 1400 DO 1401  I = 1,M                                                   LSUB0037
        JH (I) = -1                                                      LSUB0038
 1401 CONTINUE                                                           LSUB0039
C* 'VER'    CREATE INVERSE FROM 'KB' AND 'JH'     (STEP 7)               LSUB0040
 1320 VER = .TRUE.                                                       LSUB0041
      INVC = 0                                                           LSUB0042
      NUMVR   =  NUMVR  +1                                               LSUB0043
      TRIG = .FALSE.                                                     LSUB0044
      DO 1101  I = 1,M2                                                  LSUB0045
        E(I) = 0.0                                                       LSUB0046
 1101 CONTINUE                                                           LSUB0047
      MM=1                                                               LSUB0048
      DO 1113  I = 1,M                                                   LSUB0049
        E(MM) = 1.0                                                      LSUB0050
        PE(I)  =  0.0                                                    LSUB0051
        X(I) =  B(I)                                                     LSUB0052
        IF (JH(I) .NE.0) JH(I) = -1                                      LSUB0053
        MM = MM + M + 1                                                  LSUB0054
 1113 CONTINUE                                                           LSUB0055
```

```
C                 FORM INVERSE
      DO 1102  JT = 1,M
        IF (KB(JT).EQ.0)  GO TO 1102
        GO TO 600
C 600   CALL JMY
C                        CHOOSE PIVOT
 1114   TY = 0.0
        KQ = .FALSE.
        DO 1104 I = 1,M
          IF (JH(I).NE.-1.OR.ABS(Y(I)).LE.TPIV)  GO TO 1104
          IF (KQ) GO TO 1116
          IF (X(I).EQ.0.)  GO TO 1115
          IF (ABS(Y(I)/X(I)).LE.TY)  GO TO 1104
          TY = ABS(Y(I)/X(I))
          GO TO 1118
 1115     KQ = .TRUE.
          GO TO 1117
 1116     IF (X(I).NE.0..OR.ABS(Y(I)).LE.TY)  GO TO 1104
 1117     TY = ABS(Y(I))
 1118     IR = I
 1104   CONTINUE
        KB(JT) = 0
C                        TEST PIVOT
        IF (TY.LE.0.)   GO TO 1102
C                        PIVOT
        GO TO 900
C 900   CALL PIV
 1102 CONTINUE
C                 RESET ARTIFICIALS
      DO 1109  I = 1,M
        IF (JH(I).EQ.-1)  JH(I) = 0
        IF (JH(I).EQ.0)  FEAS = .FALSE.
 1109 CONTINUE
 1200 VER = .FALSE.
C                 ***         PERFORM ONE ITERATION             ***
C* 'XCK'     DETERMINE FEASIBILITY                      (STEP 1)
      NEG = .FALSE.
      IF (FEAS)  GO TO 500
      FEAS= .TRUE.
      DO 1201  I = 1,M
        IF (X(I).LT.0.0)  GO TO 1250
        IF (JH(I).EQ.0)  FEAS = .FALSE.
 1201 CONTINUE
C* 'GET'     GET APPLICABLE PRICES                      (STEP 2)
      IF (.NOT.FEAS)  GO TO 501
  500 DO 503 I = 1,M
        P(I) = PE(I)
        IF (X(I).LT.0.)  X(I) = 0.
  503 CONTINUE
      ABSC = .FALSE.
      GO TO 599
 1250 FEAS = .FALSE.
      NEG  = .TRUE.
  501 DO 504  J = 1, M
        P(J) = 0.
```

```
  504 CONTINUE                                                          LSUB0111
      ABSC = .TRUE.                                                     LSUB0112
      DO 505  I = 1,M                                                   LSUB0113
        MM = I                                                          LSUB0114
        IF (X(I).GE.0.0)  GO TO 507                                     LSUB0115
        ABSC = .FALSE.                                                  LSUB0116
        DO 508 J = 1,M                                                  LSUB0117
          P(J) = P(J) + E(MM)                                           LSUB0118
          MM = MM + M                                                   LSUB0119
  508   CONTINUE                                                        LSUB0120
        GO TO 505                                                       LSUB0121
  507   IF (JH(I).NE.0) GO TO 505                                       LSUB0122
        IF (X(I).NE.0.) ABSC = .FALSE.                                  LSUB0123
        DO 510  J = 1,M                                                 LSUB0124
          P(J) = P(J) - E(MM)                                           LSUB0125
          MM = MM + M                                                   LSUB0126
  510   CONTINUE                                                        LSUB0127
  505 CONTINUE                                                          LSUB0128
C* 'MIN'    FIND MINIMUM REDUCED COST                  (STEP 3)         LSUB0129
  599 JT = 0                                                            LSUB0130
      BB  =  0.0                                                        LSUB0131
      DO 701  J =1,N                                                    LSUB0132
        IF (KB(J).NE.0)   GO TO 701                                     LSUB0133
        DT = 0.0                                                        LSUB0134
        DO 303 I = 1,M                                                  LSUB0135
          DT = DT + P(I) * A(I,J)                                       LSUB0136
  303   CONTINUE                                                        LSUB0137
        IF (FEAS)  DT = DT + C(J)                                       LSUB0138
        IF  (ABSC)  DT = - ABS(DT)                                      LSUB0139
        IF (DT.GE.BB)  GO TO 701                                        LSUB0140
        BB = DT                                                         LSUB0141
        JT = J                                                          LSUB0142
  701 CONTINUE                                                          LSUB0143
C  TEST FOR NO PIVOT COLUMN                                             LSUB0144
      IF (JT.LE.0)  GO TO 203                                           LSUB0145
C  TEST FOR ITERATION LIMIT EXCEEDED                                    LSUB0146
      IF (ITER.GE.NCUT)  GO TO 160                                      LSUB0147
      ITER = ITER +1                                                    LSUB0148
C* 'JMY'    MULTIPLY INVERSE TIMES A(.,JT)              (STEP 4)        LSUB0149
  600 DO 610  I= 1,M                                                    LSUB0150
        Y(I) = 0.0                                                      LSUB0151
  610 CONTINUE                                                          LSUB0152
      LL = 0                                                            LSUB0153
      COST = C(JT)                                                      LSUB0154
      DO 605   I= 1,M                                                   LSUB0155
        AIJT = A(I,JT)                                                  LSUB0156
        IF (AIJT.EQ.0.) GO TO 602                                       LSUB0157
        COST = COST + AIJT * PE(I)                                      LSUB0158
        DO  606  J = 1,M                                                LSUB0159
          LL = LL + 1                                                   LSUB0160
          Y(J) = Y(J) + AIJT * E(LL)                                    LSUB0161
  606   CONTINUE                                                        LSUB0162
        GO TO 605                                                       LSUB0163
  602   LL = LL + M                                                     LSUB0164
  605 CONTINUE                                                          LSUB0165
```

```
C        COMPUTE PIVOT TOLERANCE
      YMAX = 0.0
      DO 620  I  =  1,M
         YMAX = AMAX1( ABS(Y(I)),YMAX )
  620 CONTINUE
      TPIV  =  YMAX * TEXP
C             RETURN TO INVERSION ROUTINE, IF INVERTING
      IF (VER)  GO TO 1114
C        COST TOLERANCE CONTROL
      RCOST = YMAX/BB
      IF (TRIG.AND.BB.GE.-TPIV)  GO TO 203
      TRIG = .FALSE.
      IF (BB.GE.-TPIV)  TRIG = .TRUE.
C* 'ROW'    SELECT PIVOT ROW                        (STEP 5)
C AMONG EQS. WITH X=0, FIND MAXIMUM Y  AMONG ARTIFICIALS, OR, IF NONE,
C  GET MAX POSITIVE Y(I) AMONG REALS.
      IR = 0
      AA = 0.0
      KQ = .FALSE.
      DO  1050  I =1,M
         IF (X(I).NE.0.0.OR.Y(I).LE.TPIV)  GO TO 1050
         IF (JH(I).EQ.0)  GO TO 1044
         IF (KQ) GO TO 1050
 1045    IF (Y(I).LE.AA)  GO TO 1050
         GO TO 1047
 1044    IF (KQ) GO TO 1045
         KQ = .TRUE.
 1047    AA = Y(I)
         IR = I
 1050 CONTINUE
      IF (IR.NE.0)  GO TO 1099
      AA = 1.0E+20
C             FIND MIN. PIVOT AMONG POSITIVE EQUATIONS
      DO 1010  I  = 1,M
         IF (Y(I).LE.TPIV.OR.X(I).LE.0.0.OR.Y(I)*AA.LE.X(I) ) GO TO 1010
         AA = X(I)/Y(I)
         IR = I
 1010 CONTINUE
      IF (.NOT.NEG)  GO TO 1099
C  FIND PIVOT AMONG NEGATIVE EQUATIONS, IN WHICH X/Y IS LESS THAN THE
C MINIMUM X/Y IN THE POSITIVE EQUATIONS, THAT HAS THE LARGEST ABSF(Y)
      BB = - TPIV
      DO 1030  I = 1,M
         IF (X(I).GE.0..OR.Y(I).GE.BB.OR.Y(I)*AA.GT.X(I) )  GO TO 1030
         BB = Y(I)
         IR = I
 1030 CONTINUE
C  TEST FOR NO PIVOT ROW
 1099 IF  (IR.LE.0)  GO TO 207
C* 'PIV'    PIVOT ON (IR,JT)                        (STEP 6)
      IA = JH(IR)
      IF (IA.GT.0)  KB(IA) = 0
  900 NUMPV  =  NUMPV  +  1
      JH(IR) = JT
      KB(JT) = IR
```

```
      YI = -Y(IR)
      Y(IR) = -1.0
      LL = 0
C                                   TRANSFORM INVERSE
      DO  904  J = 1,M
        L = LL + IR
        IF (E(L).NE.0.0)  GO TO 905
        LL = LL + M
        GO TO 904
  905   XY = E(L) / YI
        PE(J) = PE(J) + COST * XY
        E(L) = 0.0
        DO 906  I = 1,M
          LL  =  LL + 1
          E(LL) = E(LL)   + XY * Y(I)
  906   CONTINUE
  904 CONTINUE
C                                   TRANSFORM X
      XY  =  X(IR) / YI
      DO 908  I  = 1, M
        XOLD = X(I)
        X(I) = XOLD + XY * Y(I)
        IF (.NOT.VER.AND.X(I).LT.0..AND.XOLD.GF.0.)  X(I) = 0.
  908 CONTINUE
      Y(IR) = -YI
      X(IR) = -XY
      IF (VER) GO TO 1102
      IF (NUMPV.LE.M) GO TO 1200
C  TEST FOR INVERSION ON THIS ITERATION
      INVC  = INVC  +1
      IF  (INVC.EQ.NVER)   GO TO 1320
      GO  TO  1200
C*  END OF ALGORITHM, SET EXIT VALUES                   ***
  207 IF (.NOT.FEAS.OR.RCOST.LE.-1000.)  GO TO 203
C              INFINITE SOLUTION
      K  =  2
      GO TO 250
C         PROBLEM IS CYCLING
  160 K  =  4
      GO TO 250
C         FEASIBLE OR INFEASIBLE SOLUTION
  203 K  =  0
  250 IF (.NOT.FEAS)  K = K + 1
      DO 1399  J = 1,N
        XX  =  0.0
        KBJ  =  KB(J)
        IF (KBJ.NE.0)  XX = X(KBJ)
        KB(J) = LL
 1399 CONTINUE
      KO(1) = K
      KO(2) = ITER
      KO(3) = INVC
      KO(4) = NUMVR
      KO(5) = NUMPV
      KO(6) = JT
      RETURN
       END
```

277

277

ALGOL PROCEDURE

($*$ used in place of $\times$)

```
procedure SIMPLE(inflag,m,n,a,b,c,ko,kb,p,jh,x,y,pe,e,z);
  value m,n; real array b,p,x,y,pe[1:m],c,z[1:n],e[1:m+2],
  a[1:m,1:n]; integer array ko[1:6],jh[1:m],kb[1:n];
  integer m,n,inflag;

comment  SIMPLE solves the linear programming problem:

        find z(1), z(2), z(3),..., z(n) that

                      n
        minimizes     Σ c(j) * z(j)       subject to
                     j=1
        n
        Σ a(i,j) * z(j)  =  b(i)  for  i=1,2,3,...,m   and
       j=1

       x(1) ≥0, x(2) ≥0, x(3) ≥0,..., x(n) ≥0.

  SIMPLE simultaneously solves the dual problem:

       find w(1), w(2), w(3),..., w(m) that

                      m
       maximizes      Σ b(i) * w(i)       subject to
                     i=1
        m
        Σ a(i,j) * w(i)  ≤ c(j)  for  j=1,2,3,...,n
       i=1

       where no sign restriction is put on w.  The NEGATIVE
       of w appears in p.

  inflag, m, n, a, b, and c are input quantities:  they are
  not changed by the procedure.  The meanings of m, n, a, b,
  and c are as given in the above equations.  inflag is nor-
  mally input as a zero: this signals the procedure that no
  initial basis is provided.  If inflag is input as a non-
  zero number, the procedure will expect a basis to be pro-
  vided by setting:

        kb(j)  = 0  if column j is not in basis,
    and kb(j)  ≠ 0  if column j is in the basis.
```

After the procedure is finished, ko(1) contains a number that denotes the condition of the problem, and z and p contain the numeric answers.
If the problem is feasible and optimal, then ko(1) = 0.
If the problem admits no feasible solution, then ko(1) =1, and if the problem has an infinite solution (i.e. dual infeasible), then ko(1) = 2. If the algorithm does not terminate after 4m+10 iterations, ko(1) is set to 4 if the problem is feasible, and to 5 if yet no feasible solution has been found;

```
begin
        real aa,bb,cost,dt,rcost,texp,tpiv,ty,ymax;
        integer i,invc,ir,iter,j,jt,k,m2,mm,ncut,npiv,numvr,
        nver; Boolean absc,feas,kq,neg,trig,ver;
        iter:= 0;
        numvr:= 0;
        numpv:= 0;
        texp:= 2.↑(-16);
        ncut:= 4*m + 10·
        nver := m/2 + 5;
        m2:= m↑2;
        feas:= false;
        if (inflag ≠ 0) then go to n00;
comment    start phase one with singleton basis;
        for j:= 1 step 1 until n do
n04:      begin
          kb(j) := 0;
          kq:= false;
          for i:= 1 step 1 until m do
n05:        begin
            if (a(i,j)  = 0.) then go to n03;
            if (kq V a(i,j) < 0.) then go to n02;
            kq:= true;
n03:        end n05;
          kb(j) := 1;
n02:      end n04;
n00:    for i:= 1 step 1 until m do  jh(i) := -1;
```

```
comment      create inverse from "kb" and "jh"      (step 7);
m20:   ver:= true;
       invc:= 0;
       numvr  :=  numvr  +1;
       trig:= false;
       for i:= 1 step 1 until m2 do  e(i):= 0.0;
       mm:= 1;
       for i:= 1 step 1 until m do
k20:     begin
         e(mm):= 1.0; pe(i):= 0; x(i):= b(i);
         if (jh(i)  ≠ 0) then jh(i):= -1;
         mm:= mm + m + 1;
k13:     end k20;
       for jt:= 1 step 1 until n do
k21:     begin
         if   (kb(jt)  = 0) then go to k02;
         Get Column(jt,a,c,e,pe,y,m,cost,texp,tpiv,ymax);
         ty:= 0.0; kq:= false;
         for i:= 1 step 1 until m do
k22:       begin
           if (jh(i) ≠ -1 V abs(y(i)) ≤ tpiv) then go to k04;
           if (kq) then go to k16;
           if (x(i)  = 0) then go to k15;
           if (abs(y(i)/x(i)) ≤ ty) then go to k04;
           ty:= abs(y(i)/x(i));
           go to k18;
k15:       kq:= true;
           go to k17;
k16:       if (x(i) ≠ 0.0 V abs(y(i)) >  ty) then go to k04;
k17:       ty:= abs(y(i));
k18:       ir:= i;
k04:       end k22;
         kb(jt):= 0;
         if (ty ≤ 0.0) then go to k02;
         Pivot(ir,jt,e,jh,kb,pe,x,y,m,cost,numpv,ver);
k02:     end k21;
       for   i:= 1 step 1 until m do
         begin
         if   (jh(i)  = -1) then jh(i):= 0;
         if   (jh(i)  = 0) then feas:= false
         end;
100:   ver:= false;
comment      determine feasibility                   (step 1);
       neg:= false;
       if (feas) then go to e00;
       feas: true;
       for i:=1 step 1 until m do if(x(i)<0.) then go to 150
       else if (jh(i)  = 0) then feas:= false ;
```

```
comment     get applicable prices                              (step 2);
        if (¬feas) then go to e01;
e00:    for i:= 1 step 1 until m do
          begin
          p(i):= pe(i);
          if (x(i) < 0.0) then x(i):= 0.0
          end;
        absc:= false;
        go to e99;
150:    feas:= false;
        neg := true;
e01:    for j:= 1 step 1 until m do p(j):= 0.0;
        absc:= true;
        for i:= 1 step 1 until m do;
e11:      begin
          mm:= i;
          if (x(i) ≥ 0.0) then go to e07;
          absc:= false;
          for j:= 1 step 1 until m do
            begin
            p(j):= p(j) + e(mm);
            mm:= mm + m
            end;
          go to e05;
e07:      if (jh(i) ≠ 0) then go to e05;
          if (x(i) ≠0.) then absc:= false;
          for j:= 1 step 1 until m do
            begin
            p(j):= p(j) - e(mm);
            mm:= mm + m
            end;
e05:      end e11;
comment     find minimum reduced cost                          (step 3);
e99:    jt:= 0;
        bb := 0.0;
q02:    for j:= 1 step 1 until n do
          begin
          if (kb(j) ≠ 0) then go to q01;
          dt:= 0.0;
          for i:= 1 step 1 until m do dt:= dt + p(i)*a(i,j);
          if (feas) then dt:= dt + c(j);
          if (absc) then dt:= -abs(dt);
          if (dt ≥ bb) then go to q01 else bb:= dt;
          jt:= j;
q01:      end q02;
        if (jt ≤ 0) then go to b03;
        if (iter ≥ ncut) then go to a60;
        iter:= iter +1;
```

```
comment     multiply inverse times a(., jt)          (step 4);
f00:  Get Column(jt,a,c,e,pe,y,m,cost,texp,tpiv,ymax);
      rcost:= ymax/bb;
      if (trig ∧ bb > -tpiv) then go to b03;
      trig:= false;
      if (bb ≥ -tpiv) then trig:= true;
comment     select pivot row                         (step 5);
      ir:= 0;
      aa:= 0.0;
      kq:= false;
      for i:= 1 step 1 until m do
j02:    begin
        if (x(i) ≠ 0.0 ∨ y(i) ≤ tpiv) then go to j50;
        if (jh(i) = 0) then go to j44;
        if (kq) then go to j50;
j45:    if (y(i) ≤ aa) then go to j50;
        go to j47;
j44:    if (kq) then go to j45;
        kq:= true;
j47:    aa:= y(i);
        ir:= i;
j50:    end j02;
      if (ir ≠ 0) then go to j99;
      aa:= 10.↑20;
      for i:= 1 step 1 until m do
j03:    begin
        if (y(i)≤tpiv  ∨  x(i)≤0.0  ∨  y(i)*aa≤x(i))
        then go to j10;
        aa:= x(i)/y(i);
        ir:= i;
j10:    end j03;
      if (¬neg) then go to j99;
      bb:= -tpiv;
      for i:= 1 step 1 until m do
j04:    begin
        if (x(i)≥0. ∨ y(i)≥bb ∨ y(i)*aa>x(i)) then go to j30;
        bb:= y(i);
        ir:= i;
j30:    end j04;
j99:  if (ir = 0) then go to b07;
comment     pivot on (ir, jt)                        (step 6);
      if (jh(ir)>0) then kb(jh(ir)):= 0;
i00:  Pivot(ir,jt,e,jh,kb,pe,x,y,m,cost,numpv,ver);
      if (numpv ≤ m) then go to 100;
      invc := invc +1;
      if (invc = nver) then go to m20;
      go to 100;
```

```
comment  end of algorithm, set exit values;
b07:  if  ( ¬feas V rcost≤-1000.)  then go to b03;
comment     infinite solution;
      k :=  2;
      go to b50;
comment     problem is cycling perhaps;
a60:  k:= 4;
      go to b50;
comment     feasible or infeasible solution;
b03:  k:= 0;
b50:  if   ( ¬feas)    then k:= k + 1;
      for j:= 1 step 1 until n do z(j) := 0.0;
      for i:= 1 step 1 until m do if  (jh(i) >0)
      then z(jh(i)) := x(i);
      ko(1) := k;
      ko(2) := iter;
      ko(3) := invc;
      ko(4) := numvr;
      ko(5) := numpv;
      ko(6) := jt;
end SIMPLE;

procedure Get Column(jt,a,c,e,pe,y,m,ccst,texp,tpiv,ymax);
begin
      real aijt; integer i,j,LL;
f00:  for i:= 1 step 1 until m do y(i) := 0.0;
      LL:= 0;
      cost:= c(jt);
      for i:= 1 step 1 until m do
f11:    begin
        aijt:= a(i,jt);
        if  (aijt  = 0.)  then go to f02;
        cost:= cost + aijt * pe(i);
        for j:= 1 step 1 until m do
          begin
          LL:= LL + 1;
          y(j) := y(j)  + aijt  * e(LL)
          end;
        go to f05;
f02:    LL:= LL + m;
f05:    end f11;
      ymax:= 0.0;
      for i:= 1 step 1 until m do if  (abs(y(i))  >  ymax)
      then ymax:= abs(y(i));
      tpiv  :=  ymax * texp;
end Get Column;
```

```
procedure Pivot(ir,jt,e,jh,kb,pe,x,y,m,cost,numpv,ver);
begin
        real xy,xold,yi; integer i,j,L,LL;
i00:    numpv:= numpv + 1;
        jh(ir):= jt;
        kb(jt):= ir;
        yi:= -y(ir);
        y(ir):= -1.0;
        LL:= 0;
        for j:= 1 step 1 until m do
i01:      begin
          L:= LL + ir;
          if (e(L) ≠ 0.0) then go to i05;
          LL:= LL + m;
          go to i04;
i05:      xy:= e(L)/yi;
          pe(j):= pe(j) + cost * xy;
          e(L):= 0.0;
          for i:= 1 step 1 until m do
            begin
            LL := LL + 1;
            e(LL):= e(LL) + xy * y(i)
            end;
i04:      end i01;
        xy := x(ir) / yi;
        for i:= 1 step 1 until m do
          begin
          xold:= x(i);
          x(i):= xold + xy * y(i);
          if ( ¬ver ∧ x(i)<0. ∧ xold≥0.) then x(i):=0.
          end;
        y(ir):= -yi;
        x(ir):= -xy;
end Pivot
```

REFERENCES

1. Clasen, R. J., The Numerical Solution of the Chemical Equilibrium Problem, The RAND Corporation, RM-4345-PR, January 1965.

2. Dantzig, G. B., Linear Programming and Extensions, Princeton University Press, Princeton, New Jersey, 1963.

3. Cutler, L., and P. Wolfe, Experiments in Linear Programming: Notes on Linear Programming and Extensions--Part 63, The RAND Corporation, RM-3402-PR, February 1963.

4. International Business Machines Corporation, "IBM 7040/7044 Operating System, FORTRAN IV Language," IBM Systems Reference Library, Form C28-6329, Poughkeepsie, New York, 1963.

5. McCracken, Daniel D., A Guide to ALGOL Programming, John Wiley and Sons, Inc., New York, 1962.